UNIV-1100 — First Year Seminar: Scientific Computing Learning Community

Instructor: A. J. Meir
Peer Instructor: Lauren E. Gaines

Auburn University

November 14, 2012

Built-in Functions and Code Blocks

- Built-in functions
- Code blocks
- Boolean operators

Loops

- for
- while

Conditionals

- if
- if, else
- if, elif

Monte Carlo Integration - Revisited

Recall, if

$$I = \int_a^b f(x)dx$$

then

$$I = \text{average of } f \text{ on } (a, b) * (b - a)$$

$$I \approx (b - a)\frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

where x_i are uniformly distributed random numbers between a and b

Monte Carlo Integration - Python Program

```
def MCint():
    import random
    from math import sqrt

    n = 100000
    s = 0
    for i in range(n):
        x = random.uniform(0,1)
        s += sqrt(1.0 - x**2)
    I = (float(1 - 0)/n)*s
    print I
MCint()
```

Monte Carlo Integration - A Better Program

```
def f1(x):
    from math import sqrt
    return sqrt(1 - x**2)

def MCint(f,a,b,n):
    import random

    s = 0
    for i in range(n):
        x = random.uniform(a,b)
        s += f(x)
    I = float(b - a)/n*s
    return I
```

Monte Carlo Integration - A Better Program (continued)

```
a = 0
b = 1
n = 1000000

I = 4*MCint(f1,a,b,n)

print I
```

Monte Carlo Integration - Creating Modules

- We have used modules
- A module is a collection of useful data and functions
- Functions in a module can be reused in different programs
- If you have some general functions that can used in more than one program, consider making a module
- Making modules is easy: just collect functions in a file, and you have a module

Monte Carlo Integration - Creating Modules

Create a module *functions* (which provides various functions), that is create a file functions.py (which contains definitions of various functions).

```
from math import sqrt

def f1(x):
    return sqrt(1 - x**2)

def f2(x):
    return 1/x
```

Monte Carlo Integration - Creating Modules

Create a module *MCinteg* which provides the function MCint, that is create a file MCinteg.py (which contains the definition of MCint).

```
def MCint(f,a,b,n):
    import random

    s = 0
    for i in range(n):
        x = random.uniform(a,b)
        s += f(x)
    I = float(b - a)/n*s

    return I
```

Monte Carlo Integration - Creating Modules

Putting it all together

Import the modules you created and run your program

```
import functions
import MCinteg

a = 0
b = 1
n = 1000000

I = MCinteg.MCint(functions.f1,a,b,n)

print 4*I
```

Trapezoidal Rule

We can approximate the integral

$$I = \int_a^b f(x)dx$$

as follows. For some n set $h = \frac{b-a}{n}$ and compute the approximation

$$I \approx h\left[\frac{f(a)}{2} + \sum_{i=1}^{n-1} f(a+ih) + \frac{f(b)}{2}\right]$$

Trapezoidal Rule - The Program

```
def Trapezoid(f,a,b,n):

    h = (b - a)/float(n)
    s = f(a)/2.0

    for i in range(1,n):
        s += f(a + i*h)

    s+= f(b)/2.0

    I = s*h

    return I
```

Using the Trapezoidal Rule to Approximate π

```
import functions
import Integ

a = 0
b = 1
n = 1000

I = Integ.Trapezoid(functions.f1,a,b,n)

print 4*I
```

Vectors and Vectorized Operations

Python (actually numpy) allows dealing with arrays or vectors

- We can perform array computations, or vector operations (called vectorization
- Array computations are useful for more than plotting curves
- Useful when we need to compute with large amounts of numbers, we store the numbers in arrays and compute with arrays, giving shorter and faster code

Vectors and Vectorized Operations

- In general a vector $\mathbf{v}$ is an n-tuple of numbers $\mathbf{v} = (v_0, v_1, v_2, \ldots, v_{n-1})$
- We can do various mathematical operations on vectors
- Vectors can be represented by lists, v_i is stored as `v[i]`
- Arrays are generalizations of vectors, these have multiple indices, e.g., matrices
- The number of indices in an array is the rank, or number of dimensions
- A vector is a one-dimensional, or rank 1 array
- We use Numerical Python arrays instead of lists to represent mathematical arrays (this is computationally more efficient)

Plotting and Graphing

To graph a function f, we evaluate the function at points (in its domain).

This yields points along the curve $y = f(x)$, which we can store in one dimensional arrays **x** and **y**

To obtain a graph we connect these points with line segments, this is done programatically with $\texttt{plot}(\texttt{x}, \texttt{y})$

Plotting and Graphing

```
import numpy
import matplotlib.pyplot as plt

t = numpy.linspace(0,3,51)
y = t**2*numpy.exp(-t**2)

plt.plot(t,y)
plt.savefig('img1.png')
plt.savefig('img1.pdf')
plt.show()
```

Plotting and Graphing